

# Self-Healing Reconfigurable Logic using Autonomous Group Testing

CARTHIK A. SHARMA AND RONALD F. DEMARA

University of Central Florida

and

ALIREZA SARVI

Xilinx Inc.

---

A self-healing, self-organizing evolvable hardware system is developed using SRAM-based reconfigurable *Field Programmable Gate Arrays (FPGAs)* and group testing principles. It employs adaptive group testing techniques to autonomously maintain resource viability information as an organic means of transient and permanent fault resolution. Reconfigurability of the SRAM-based FPGA is leveraged to identify logic resource faults which are successively excluded by group testing using alternate device configurations. This simplifies the system architect's role to definition of functionality using a high-level Hardware Description Language (HDL) and system-level performance versus availability operating point. System availability, throughput, and mean time to isolate faults are monitored and maintained using an Observer-Controller model. Dedicated test vectors are unnecessary as the algorithm operates on the output response produced for real-time operational inputs. Results are demonstrated using a Data Encryption Standard (DES) core that occupies approximately 305 FPGA slices on a Xilinx Virtex-II Pro FPGA. With a single simulated stuck-at-fault, the system identifies a completely validated replacement configuration within three to five positive tests. Results also include approaches for optimizing population size, resource redundancy, and availability. The approach demonstrates a readily-implemented yet robust organic hardware application framework featuring a high degree of autonomous self-control.

Categories and Subject Descriptors: B.8.1 [**Hardware**]: Performance and Reliability - *Reliability, Testing, and Fault-Tolerance*; C.1.3 [**Computer Systems Organization**]: Other Architecture Styles - *Adaptable Architectures*; I.1.2 [**Computing Methodologies**]: Symbolic and Algebraic Manipulation - Algorithms - *Non-Algebraic Algorithms*

General Terms: Autonomous Systems, Group Testing, Reconfigurable Architectures

Additional Key Words and Phrases: Evolvable Hardware, Reliable Systems, Organic Computing

---

This research was supported in part by NASA Intelligent Systems NRA Contract NNA04CL07A.

Authors' addresses: Carthik A. Sharma, Ronald F. DeMara, School of Electrical Engineering and Computer Science, University of Central Florida, Box 162362, Orlando, FL 32816-2362; Alireza Sarvi, 2100 Logic Drive, San Jose, CA 95124.

Permission to make digital/hard copy of part of this work for personal or classroom use is granted without fee provided that the copies are not made or distributed for profit or commercial advantage, the copyright notice, the title of the publication, and its date of appear, and notice is given that copying is by permission of the ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee.

© 2007 ACM

## 1. INTRODUCTION

Improving the fault tolerance of reconfigurable devices is increasingly important in domains ranging from mission critical embedded applications [Yui et al. 2003][Perotti et al. 2002] [Kowalski et al. 2005] to the use of FPGAs in physically remote environments such as deep space probes and satellites [Katz and Some 2003][Bar-sever et al 2003][Fischman *et al* 2004]. Mission sustainability realized by autonomous repair of

reconfigurable devices such as FPGAs is of particular interest to both in-flight applications and ground support equipment for space missions at NASA. For in-flight applications, these devices encounter harsh environments of mechanical/acoustical stress during launch, high ionizing radiation, and thermal stress [Srouer and MacGarrity 1988]. NASA terrestrial applications such as ground-support systems also routinely employ FPGAs extensively for tasks ranging from launch control to signal processing. These applications such as data acquisition devices and instrumentation systems seek to incorporate self-healing capabilities and provide extended calibration cycles. The *Advanced Data Acquisition System (ADAS)* [Perotti et al. 2002] is a signal acquisition and processing system for launch control measurements that is typical of real-time NASA applications that heavily utilize FPGAs and have high reliability, availability, and maintainability requirements. Ionization, electromigration, hot carrier effects, and other device degenerative effects may cause device faults in the FPGAs used by such applications. In all of the above scenarios, these devices are mandated to operate reliably for long mission durations with limited or absent capabilities for diagnosis/replacement and little onboard capacity for spares.

### 1.1 Organic and Autonomous Systems

Increasing resource constraints and system complexity have led NASA and others towards autonomous adaptive and organic systems. NASA has identified autonomous systems as a key area for research [Clancy 2002] and research is being conducted on building self managing applications and systems [Truszkowski et al. 2004][Wyatt et al. 1999]. *Autonomic Computing* [Sterritt 2005] defines the vision for self-managed *Organic Systems*. Such systems supplant human monitoring and repair requirements with *self-organization* and *self-healing* properties. Additionally they reduce system complexity by reducing the designer's task to defining high-level system goals. The system achieves these goals through *self-optimization* by virtue of its *self-monitoring* and *self-adjusting* attributes. These attributes can be realized using an *Observer-Controller* paradigm [Richter et al. 2006].

### 1.2 FPGA Fault Tolerance Techniques

As shown in Fig. 1, all fault tolerance approaches involve *fault diagnosis* and *fault recovery strategies*. Current FPGA fault tolerance approaches [Cheatham et al. 2006] can be broadly classified into *Device Level* approaches and *Configuration Level*

approaches. Device level schemes rely on spare modules [D’Angelo et al. 1998] [Kastensmidt et al. 2005][Li et al 2000] or spare resources of varying granularity [Xu et al 1999][Lach et al. 1998]. Configuration level approaches [Garvie and Thompson 2004][Vigander 2001][Keymeulen et al 2000] treat the FPGA as a set of abstract resources from which fault-free resources are used to place and route circuits. Additionally, as shown in Fig. 1, these circuit configurations can be created either at design-time or at runtime.

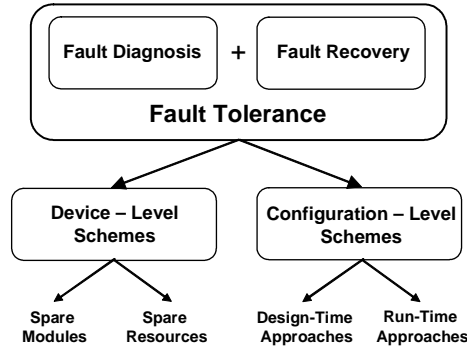


Fig. 1. Overview of FPGA Fault Tolerance Schemes

*Fault diagnosis* typically consists of *fault detection* [Mitra and McCluskey 2000][DeMara and Sharma 2005] and *fault isolation* [Abramovici et al. 2001][Kahng and Reda 2004]. The *fault recovery* phase then utilizes the results from the diagnosis phase to realize fault tolerance. A majority of FPGA fault diagnosis methods use *resource-oriented tests* [Abramovici et al. 2001], or *functional testing* of the system [Vigander 2001][Keymeulen et al 2000].

A major limitation of exhaustive resource-based testing strategies is that the device has to be taken offline for testing to be completed, thus reducing availability. An alternative to exhaustive resource testing is functional system testing as employed by some configuration-based fault tolerance approaches. However, functional testing schemes can only detect faults at the system-level, as they fail to isolate individual faulty resources. *Autonomous Group Testing (AGT)* provides a novel method to isolate faults at progressively increasing granularity using only functional testing.

### 1.3 Group Testing

*Group Testing* provides an efficient alternative to exhaustive testing. The theory and practice of group testing is described in Du and Hwang [Du and Hwang 2000]. Group testing was first introduced by Dorfman [Dorfman 1943] whose work dealt with testing a

large set of blood samples taken from World War II recruits. To avoid the cost of testing the samples individually, subsets of these samples were pooled for testing. If a pool tested positive, then the samples that contributed to the pool were used to create smaller pools, until the number of suspect samples in a pool was small enough to allow individual tests. This introduced the idea of using group testing when the number of defectives in a sample space is small compared to the total number of samples. Subsequent research has identified group testing-based solutions for problems as diverse as DNA library scanning [Barillot et al. 1991] and detecting shorts in electrical circuits [Garey et al. 1976].

The fundamental problem in group testing is to identify the subset of defective members,  $\mathbf{D} \subset \mathbf{R}$ , where  $\mathbf{R}$  is the set of all resources, using a minimal number of tests,  $t$  on  $g$  groups or subsets of  $\mathbf{R}$ . A group testing algorithm is *adaptive* if the tests are conducted in *stages* where the tests in later stages are based on the results of previous stages, and *non-adaptive* otherwise. In *combinatorial* group testing, the number of defectives is a constant, as opposed to *probabilistic* group testing, where the probability of each member being defective is known. The FPGA fault isolation problem maps to the class of adaptive combinatorial group testing algorithms.

The problem of isolating a fault resource in the FPGA has several similarities to the *counterfeit coin problem* [Smith 1947]. As originally stated, the problem consists of identifying one counterfeit coin from among twelve coins using no more than three weighings with a balance. The counterfeit coin has a weight different from the genuine coins. As adapted to FPGA fault isolation, the problem is to identify faulty resources with the knowledge that the use of a faulty resource in creating a functional configuration may cause discrepant outputs. As in the counterfeit coin problem, configurations describe collections of resources that are “weighed” against a known-good configuration, by comparing their outputs.

The proposed AGT-based fault tolerance paradigm leverages device reconfigurability to realize a self-adaptive, organic hardware platform for the implementation of combinational digital designs. This facilitates a fault tolerance system that is adaptive, allowing for continuous trade-offs between system-level goals, unlike previous approaches. However, the realization of existing methods in conjunction with commercially available design tools, on commercial FPGAs is a major challenge. This fault isolation technique is extended to provide an organic system based on the *Observer-Controller* paradigm [Richter et al. 2006] that adapts to meet system-level performance requirements. Functionally equivalent designs that utilize differing subsets of resources

called *configurations* are created which also act as the test-groups. These designs test the resources comprising the reconfigurable architecture without the use of special test vectors or coding schemes. The system architect's role is simplified to defining the required functionality using a high-level *Hardware Description Language (HDL)*. These system goals, such as maintaining availability and goodput, dictate the priorities at the time of operation. The system *goodput* is defined as the percentage of correct, useful outputs produced, in a manner similar to the goodput of networks [Floyd and Fall 1999]. System availability, goodput, and mean time to fault isolation are continually monitored and maintained by using an *Observer-Controller* model.

The rest of this paper is structured as follows: Section 2 compares previous approaches in FPGA fault tolerance, highlighting the innovations presented here. Section 3 presents the mathematical model of the paradigm as also the details of the AGT algorithm. Section 4 describes hardware implementation details for a Xilinx Virtex-II Pro FPGA. Section 5 shows the results from experiments demonstrating successful realization of expected performance, and finally, Section 6 discusses emerging challenges and opportunities offered by Xilinx, as also possible extensions of the AGT self-healing hardware paradigm.

## 2. PREVIOUS WORK

The fault diagnosis and resolution techniques used by previous FPGA fault tolerance methods provide the basis for their classification. The granularity and precision with which faults are isolated vary in these methods – ranging from methods that only require fault detection during fault diagnosis, to methods that isolate the fault to increasing levels of granularity. Early fault resolution methods relied on coarse-grained system-level redundancy. Current research is focused on fault resolution by means of alternate configurations that avoid the use of the faulty resource. These alternate configurations are either created at design-time [Huang and McCluskey 2001][Keymeulen et al 2000] or at run-time [Vigander 2001] [Garvie and Thompson 2004] after a failure is detected.

### 2.1 Fault Detection and Location using Functional and Exhaustive Testing Techniques

Fig. 2 shows the classification of fault diagnosis methods based on the approach to testing. Fault diagnosis methods consist of a fault detection method and an optional fault location or isolation technique. Coarse-grained hardware redundancy approaches such as

TMR rely on *functional testing* using a voting element. The voting element identifies the faulty module based on a majority vote on the output response of the functional implementation. A voter is also used by the evolutionary repair mechanisms proposed by Vigander [Vigander 2000] and Garvie et al. [Garvie and Thompson 2004]. As mentioned in Garvie et al., such systems rely on a voting element that is assumed unsusceptible to failure. In addition, as summarized in Table I, voting elements used in these approaches cannot locate faults at a finer granularity than the module in which the fault occurs.

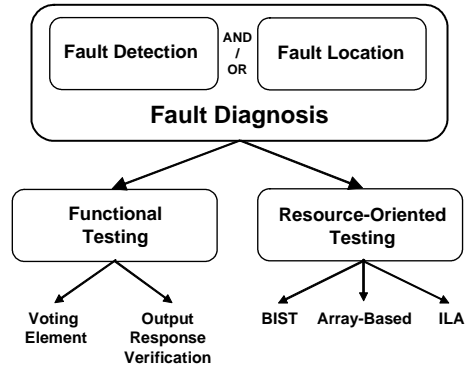


Fig. 2. Overview of FPGA Fault Diagnosis Techniques

As shown in Fig. 2., *Resource-oriented testing* techniques include *Built In Self Test* (BIST,) *Iterative Logic Array* (ILA) testing and *Array Based* testing as discussed by Doumar and Ito [Doumar and Ito 2003]. These techniques consist of subjecting the resources under test to an exhaustive battery of serialized test inputs. Of the resource-oriented testing methods, BIST [Stroud et al. 1997] is widely favored by current fault recovery methods such as the roving STARS approach [Abramovici et al. 2000], which realizes fault location and exhaustive testing of the logic resources. As listed in Table 1, a major limitation of the roving STARS technique is that unlike other methods, the detection latency is very high due to the sequential roving nature of the area under test. A majority of the methods in Table 1, such as methods 2, 3, 4 and 5 rely on exhaustive serial testing procedures that utilize either the resources, or the entire input space of the function to evaluate the fitness of the FPGA resources. In contrast to these, the proposed *Autonomous Group Testing* (AGT) technique realizes logic resource testing using functional tests on the system. It does so without exhaustive functional or resource-based testing, relying only on the runtime output responses of the FPGA.

A major limitation of BIST is that the portion of the FPGA undergoing the test has to be taken offline. In BIST-based isolation, the ability to test groups of resource of varying

sizes is essential. Recently combinatorial group testing techniques have begun to be applied to BIST based diagnosis [Kahng and Reda 2004][Kahng and Reda 2006]. Fault location with very high precision is achieved by integrating group testing methods such as *batching*, *digging*, *jumping*, and *doubling* with scan-based BIST on selected benchmark circuits. A drawback is that the FPGA has to be offline and dedicated to testing, making this method suitable for post-manufacturing defect testing. Moreover, a comprehensive fault tolerance solution is needed. Previous fault tolerance approaches assume that the required high-precision fault location techniques are available prior to reconfiguration as stated in [Huang and McCluskey 2001]. However, as listed in Table 1, this method does not locate the fault, and cannot be easily integrated into a design-flow that accommodates COTS development tools. Keymeulen describes a technique that creates configurations based on populational fault tolerance at design time. From Table 1, it can be seen that this method relies on output response analysis which does not locate faults, and also fails to distinguish between transient and permanent faults.

Table I shows that except for the roving STARS approach, Garvie’s *scrubbing and jiggling* method and the proposed technique, all other methods either cannot tolerate transient faults, or suffer penalties that arise from not being able to distinguish transient faults from permanent faults. Lastly, only methods 1, 4, 5, and 6 have been demonstrated by means of a device-specific prototype. The proposed AGT-based approach uses post-place-and-route simulation executables created using the Xilinx ISE 9.1i tools to demonstrate viability. The same post-place-and-route model can also be used without any modifications to create bitstreams that configure the FPGA hardware.

Table I. Fault Diagnosis Capabilities and Limitations for Selected Approaches

	Online Fault Detection	Resource-Level Fault Location	Avoids Exhaustive Serial Test Procedure	COTS Design Flow Capable	Transient Fault Coverage	Non-functional Test Inputs Avoided	Low Detection Latency	Device-Specific Prototype
1. TMR	✓		✓	✓			✓	✓
2. Garvie	✓				✓	✓	✓	
3. Vigander	✓						✓	
4. Keymeulen						✓	✓	✓
5. STARS	✓	✓			✓			✓
6. Huang	✓		✓				✓	✓
<b>7. AGT</b>	✓	✓	✓	✓	✓	✓	✓	✓

## 2.2 Fault Tolerance using Device and Configuration Based Approaches

Fault recovery schemes for FPGAs can be classified on the basis of the type of redundancy leveraged. Fig. 3 shows an overview of these pre-defined hardware redundancy-based techniques and methods that leverage functionally equivalent redundant configurations. *Device redundancy-based* fault tolerance strategies utilize a predetermined set of physical resources as hot or cold spares at some fixed granularity. They supply a prearranged replacement after a fault is detected. The *Module-Based* techniques shown in Fig. 3 which rely on redundant hardware modules, such as *Triple Modular Redundancy (TMR)*, are among the most commonly used approaches to tolerate permanent faults [Siewiorek and Swarz 92]. Adaptations of the TMR scheme to FPGAs include [D'Angelo et al. 1998][Kastensmidt et al. 2005][Li et al. 2000] and [Vigander 2000]. TMR employs a fixed pool of three identical datapaths to compute each function simultaneously in triplicate. Under TMR, only the majority vote of the three outputs is propagated. This supports the masking of a single faulty module for a given input vector. However, multiple faulty modules can leave the system unprotected as the quantity of spares in the resource pool is fixed. In addition, this incurs extra weight, space, and power at a resolution that may be excessively large or small relative to the fault.

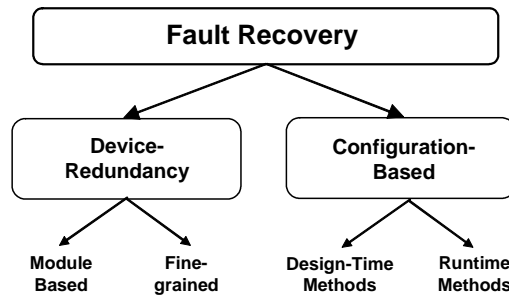


Fig. 3. Overview of FPGA Fault Recovery Schemes

Instead of providing resource redundancy at the module level, fine-grained approaches provide redundant logic resources such as in the tile-based precompiled configuration approach [Lach et al. 1998]. The tiles are atomic fault-tolerant logic blocks that consists of two or more configurable logic blocks that cover each other in case of a fault. Spare tiles can be selected when needed, but their functionality is predetermined and thus limited. On the other hand, STARS [Abramovici et al. 2001] is a resource-oriented diagnostic test approach that performs BIST-based tests on roving sub-sections of the FPGA. Portions are continually taken offline in succession and tested while the

functionality is moved to a new location. STARS' detection latency can be excessive since the tests must sweep through all resources. Also, STARS' power consumption and unavailability due to unnecessary reconfigurations when no faults have yet occurred can be prohibitive. Device redundancy methods have a high area and device overhead and their fault tolerance is limited by the pre-determined redundancy.

*Configuration-based* approaches rely on a population of alternate configurations that each use a different set of logical resources to respond to faults. These can be created either at *design-time*, or at *runtime*, after the fault has occurred. The pre-compiled configuration based technique [Huang and McCluskey 2001] creates alternative configurations at design time that use different equivalent columns of FPGA resources. In their non-overlapping scheme, which has the least resource overhead, a total of  $C(k+m, m) = (k+m)! / (m!k!)$  configurations are required to tolerate faults in  $m$  columns, where  $k$  is the number of columns in the base configuration. The required design-time effort for this approach is high, as it requires manual modification of the design to fit into column sets. Also, the number of horizontal routes available to the designer is reduced by the resources consumed by the approach. The fitness-based and population-based evolutionary hardware approaches for *Field Programmable Transistor Arrays (FPTAs)* proposed by Keymeulen et al. [Keymeulen et al. 2000] creates alternative configurations for anticipated faults and at runtime for observed faults respectively. This method provides good resource coverage and passive runtime operation, however system uptime is impacted severely by failure occurrence. Also, additional external computational capacity is required to implement the genetic algorithm that creates the population-based solution at runtime.

Another approach that uses evolutionary principles is the GA-enhanced TMR proposed by Vigander [Vigander 2001]. In this approach, TMR is extended to utilize faulty FPGAs that have been partially regenerated using evolutionary algorithms. He demonstrates that FPGA-based implementations of 4-bit x 4-bit multipliers can be automatically reconfigured to realize partial refurbishment. Since each partially refurbished multiplier is deficient with respect to only certain input pairs, a voting arrangement of partially refurbished parts exhibits complete regeneration of functionality. Garvie et al.'s method [Garvie et al. 2004] tolerates permanent faults using jiggling. Jiggling involves repairing a faulty configuration by using an evolutionary algorithm that uses the other two healthy modules and fitness feedback from the TMR voting element. Vigander's, Garvie's and other *n*-plex *spatial voting* approaches [Milliord et al. 05]

deliver real-time fault resolution, but increase power consumption and area requirement  $n$ -fold during fault-free operation. Previously, these evolutionary approaches have only been simulated using hypothetical device models. They did not attempt application to *Commercial Off The Shelf (COTS)* FPGAs and development tools.

### 2.3 Improving Reliability of Redundant Systems Using Autonomous Group Testing

In state-of-the-art Xilinx SRAM-based FPGAs, the device configuration can be modified without interrupting the normal operation of the device. For space applications, it is typical to perform such *configuration scrubbing* periodically to repair any configuration errors due to *Single Event Upsets (SEUs)* [Bridgford et al. 2007]. The Xilinx TMR tool software [Carmichael 2006] can be used to not only triplicate the user's design, but also insert logic to repair transient user memory errors and upsets due to SEUs. TMR can be combined with the scrubbing method to have a reliable system while preventing soft errors. However, configuration scrubbing only refreshes a single complete configuration and therefore cannot be used to address permanent faults [Carmichael et al. 2000]. While a  $n$ -modular redundancy scheme such as TMR ensures validated correct output, the proposed AGT-based technique can minimize the risk of having two faulty modules. The comparators of the Xilinx TMR tools can be used to detect the discrepancy among the redundant modules. Discrepancies reported by the comparators can be used to target all resources used by a faulty module. Once the faulty module is identified, the AGT-based algorithm can localize the fault to a logic slice. AGT aims to avoid system failure by providing methods to isolate permanent faults and maintain a healthy population of configurations for each redundant module.

### 2.4 Summary

A common characteristic of the previous fault tolerance approaches is that some, such as approaches 1, 2, 3 and 4 in Table I, detect the fault and recover from the fault without locating the faulty resource precisely. Others, such as approach 6 in Table I, assume that fault diagnosis is complete before the fault recovery process commences. In the former cases, the fault recovery solution could benefit from precise knowledge regarding the location of the fault. In the latter case, fault recovery cannot proceed until the fault is precisely detected or located. The proposed approach resolves this dichotomy by providing precise fault location and low-latency fault response. An additional

improvement is that methods to control the mean time to scour faults and maintain system availability are provided that make this an autonomous, adaptive fault tolerance solution. In addition, as shown in Table I, unlike most of the previous evolutionary approaches, Xilinx 9.1i COTS software for Virtex II Pro FPGAs is used to demonstrate results for the AGT approach.

### 3. GROUP TESTING-BASED ISOLATION

The logic resources on a Xilinx FPGA device are organized as a two-dimensional array of *Configurable Logic Blocks* (CLBs) [Xilinx 2006]. Each CLB consists of 4 *slices*, which in turn contain two 4-input *Look Up Tables* (LUTs). In the AGT-based fault isolation method described, a *logic resource* refers to a slice in the FPGA. As shown in Fig. 4, the FPGA is seen as a two-dimensional array of resources, each resource being a slice. The fault model accounts for stuck-at faults at the inputs of one of the two LUTs in a slice specified by its  $(x,y)$  coordinate pair.

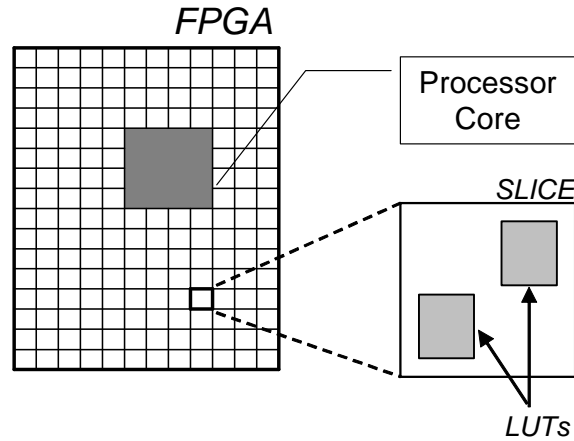


Fig. 4. FPGA Resources as Seen by the Group Testing Algorithm

#### 3.1 Terminology and Nomenclature

Let  $\mathbf{R}$  denote the set of all resources  $r_i(x,y) \in \mathbf{R}$  under test as specified by their  $(x,y)$  coordinates. A set of functionally-equivalent logic configurations,  $\mathbf{C}$ , consisting of subsets  $\mathbf{c}_i, 0 \leq i \leq p$ , where  $p$  quantifies the size of a *population of design configurations*. Each configuration realizes the combinatorial logic required for the application.

The *population preset* value  $p_{preset}$  determines the maximum number of individuals in a generation so that  $p_{stage} \leq p_{preset}$  as testing progresses. At each *stage* of the adaptive

testing algorithm, the configurations in the population are replaced by new designs, creating a new *generation* of individuals.

$\mathbf{T}$  denotes the set of binary input vectors applied and  $t_i \in \mathbf{T}$  are the individual input vectors. These inputs to the implemented combinatorial logic are also the test vectors for the isolation procedure. Let the function implemented on the FPGA be denoted by  $F(\mathbf{T}, \mathbf{c}_i)$ . If any of the resources in  $\mathbf{c}_i$  used to realize  $F(\mathbf{T}, \mathbf{c}_i)$  are faulty, then the response will deviate from the correct realization, for some subset  $\mathbf{T}' \subset \mathbf{T}$  which articulate the fault as follows:

**Definition 1.** The *syndrome*  $\mathbf{T}'$  of a configuration  $\mathbf{c}_i$  is the set of positive tests for the configuration.

**Definition 2.** The *discrepancy function*  $D(\mathbf{T}', \mathbf{c}_j)$  yields a set of all outputs that are not equal to the correct output, as realized when tests comprising the syndrome,  $\mathbf{T}'$  are applied to configuration  $\mathbf{c}_j$ . Tests  $\mathbf{T}' \subset \mathbf{T}$  on a subset  $\mathbf{c}_j$  are *positive* if and only if  $D(\mathbf{T}', \mathbf{c}_j) \neq \{\}$ , and *negative* otherwise.

**Definition 3.** The *articulation rate*  $a(\mathbf{c}_i)$  for a configuration  $\mathbf{c}_i$  is the ratio of the number of incorrect outputs to the cardinality of the entire output space:

$$\text{Articulation rate, } a(\mathbf{c}_i) = \frac{|\mathbf{T}'|}{|\mathbf{T}|}. \quad (1)$$

Since the articulation rate cannot be controlled by the designer, it introduces randomness into the rate of progress of fault isolation as discussed in section 3.6. Fault isolation proceeds by reducing the cardinality of the set of *suspects*,  $\mathbf{S}$ .  $\mathbf{S}$  is defined as the intersection of resources  $r_i(x,y) \in \mathbf{c}_i$  used by all  $\mathbf{c}_i \forall D(\mathbf{T}', \mathbf{c}_i) \neq \{\}$ . The set of all viable resources tenable to creating fault-free configurations is denoted by  $\overline{\mathbf{S}}$ , such that  $\mathbf{S} \cup \overline{\mathbf{S}} = \mathbf{R}$ .

**Definition 4.** *Forward Progress* is made, if, as fault isolation proceeds,  $|\mathbf{S}|$  decreases and  $|\overline{\mathbf{S}}|$  decreases, until finally  $|\mathbf{S}| = d$ , the number of known defectives.

As fault isolation progresses  $|\mathbf{S}|$  decreases and  $|\overline{\mathbf{S}}|$  increases, until finally  $|\mathbf{S}| = d$ , the number of known defectives.

**Definition 5.** The *defect scouring ratio*  $d(\text{stage})$  defines the ratio of number of known suspects  $|\mathbf{S}|$  to  $|\overline{\mathbf{S}}|$ , given the number of test stages that have been completed:

$$d(\text{stage}) = \frac{|\mathbf{S}|}{|\overline{\mathbf{S}}|}. \quad (2)$$

### 3.2 AGT Algorithm Overview

As shown in Fig. 5, the AGT algorithm comprises of three phases of fault isolation which occur after the fault has been detected. First, in the initialization phase, all elements of the History Matrix,  $H$ , described in Section 3.3, are initialized to zero. In addition, since the isolation procedure is yet to begin, the set of suspect resources,  $S$  is equal to the set of resources under test,  $R$ . After initialization, the  $p_{stage}$  configurations that comprise the first testing stage are created, which forms the second phase of the algorithm. The third phase consists of performing tests on the configuration thus created. Phases 2 and 3 are repeated until the defective resource is isolated.

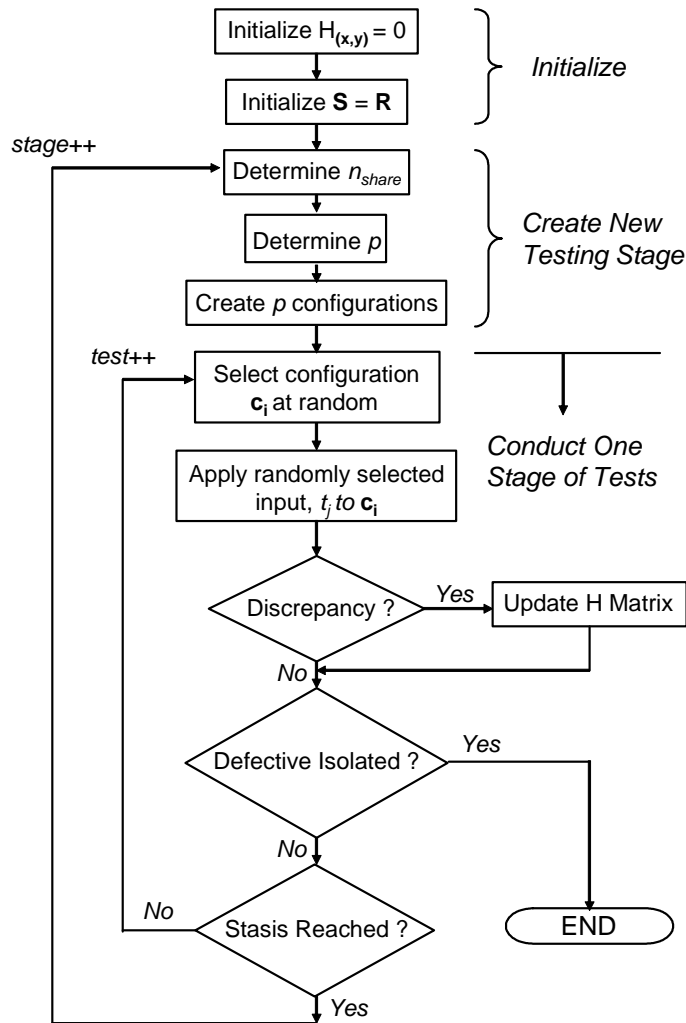


Fig. 5. AGT Process Flow

Before the configurations for a stage are created in phase 2, the *equal share factor*,  $n_{share}$ , and the population size,  $p_{stage}$ , are determined as described in Sections 3.4 and 3.5, respectively. Once  $n_{share}$  is known, the  $p_{stage}$  individuals that comprise the first test stage are created using the *Fault Injection and Analysis Toolkit (FIAT)* described in Section 4. During the fault isolation phase shown in Fig. 5, isolation proceeds by applying random test vectors which emulate the input data stream to randomly selected configurations that comprise the first test stage. This process continues until *stasis* is attained, as described in Section 3.6. After the system attains stasis, a new testing stage is created, and fault isolation is pursued until the defective resource is identified.

### 3.3 Tracking Defectives Using a History Matrix

The history matrix,  $H$ , keeps track of the discrepancy counts of the resources. As shown in Fig. 5, all elements in the  $H$  matrix are initialized to zero. As a stage of tests proceeds, for each test  $t_i$  for which  $\mathbf{D}(t_i, \mathbf{c}_j) \neq \{\}$ , all  $H$  matrix entries  $H_{(x,y)}$  are incremented by one where  $(x,y)$  are the coordinates of all  $r_i(x,y) \in \mathbf{c}_j$ . Over time, the maximal elements in  $H$  identify suspect resources by their coordinates. Under a single-fault assumption, fault isolation is complete when a unique maximum can be identified in  $H$ . The defective resource will be identified by the coordinates of the maximal element in  $H$ .

### 3.4 Test Group Formation Based on Equal Sharing

Initially,  $\mathbf{S} = \mathbf{R}$ , since no information is available regarding the fitness of any of the resources. The algorithm proceeds in stages, with a new generation of individuals being created in each stage. In each stage, the members of  $\mathbf{S}$  are equally shared among the configurations  $\mathbf{c}_i$ ,  $0 < i < p_{stage}-1$  in the generation.

The remaining  $n_{reqd}$  resources required to realize the design are randomly selected from the set  $\overline{\mathbf{S}}$  which has a cardinality  $|\mathbf{R}| - |\mathbf{S}|$ . Thus each individual  $\mathbf{c}_i$  will be allocated  $|\mathbf{R}| - |\mathbf{S}| + |\mathbf{S}|/p_{stage}$  resources. Hence if the number of suspects  $|\mathbf{S}|$  is small enough such that  $|\mathbf{R}| - |\mathbf{S}| + |\mathbf{S}|/p_{stage} > n_{reqd}$ , then the configurations in that group will have mutually exclusive shares of the suspect resources, with each individual configuration  $\mathbf{c}_i$  being allocated exclusive resources  $r_j(x,y) \in \mathbf{c}_i$ ,  $r_j(x,y) \notin \mathbf{c}_k$ , where  $0 < k < p_{stage}-1$ . Otherwise, some suspect resources need to be shared among the configurations to meet the application resource demand  $n_{reqd}$ . The maximum cardinality of  $|\mathbf{S}|$  such that mutually exclusive shares of suspect resources are possible, denoted by  $|\mathbf{S}_{max}|$  can be obtained by evaluating the following expression:

$$|R| - |S_{\max}| + \frac{|S_{\max}|}{p_{\text{preset}}} = n_{\text{reqd}}, \text{ which yields:}$$

$$|S_{\max}| = \frac{p_{\text{preset}}}{(1 - p_{\text{preset}})} \times (n_{\text{reqd}} - |R|) \quad (4)$$

If  $|S| > |S_{\max}|$  then the equal share factor,  $n_{\text{share}}$ , by rearranging Eq. 5 to yield Eq. 6:

$$n_{\text{reqd}} = |R| - |S| + n_{\text{share}} \quad (5)$$

$$n_{\text{share}} = n_{\text{reqd}} - |R| + |S| \quad (6)$$

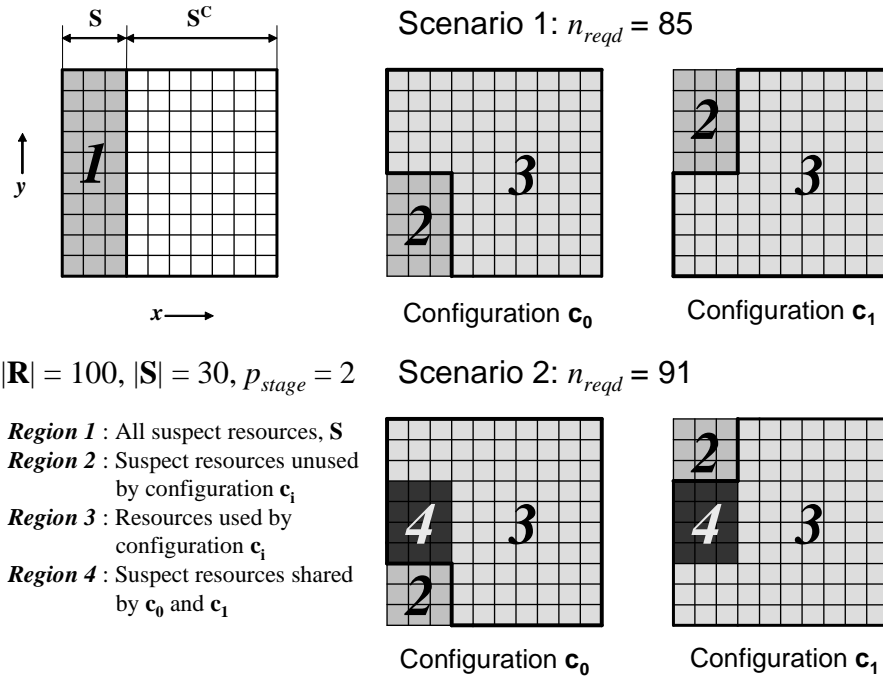


Fig. 6. Sharing the Suspect Resources Equally – Two Different Scenarios

Fig. 6 shows an example of how  $|S| = 30$  suspect resources from among  $|R| = 100$  resources are shared among  $p_{\text{stage}} = 2$  configurations. In case 1,  $n_{\text{reqd}} = 85$ , yielding  $|S_{\max}| = 30$  using Eqn. 4. Since  $|S| = |S_{\max}|$  in this scenario, configurations  $c_0$  and  $c_1$  use mutually exclusive subsets of  $S$ , and they both use all  $r_i(x,y) \in \bar{S}$  to satisfy the application resource demand. In scenario 2, however,  $n_{\text{reqd}} = 91$ , and thus,  $|S_{\max}| = 18$ . Since  $|S| > |S_{\max}|$ , the equal share factor is evaluated using Eqn. 6 to be  $n_{\text{share}} = 21$ . As shown for case 2 in Fig. 5,  $c_0$  and  $c_1$  share  $|S| - 2 \times n_{\text{share}} = 12$  suspect resources.

### 3.5 Population Size Adaptation

In order to reduce the number of individuals under test, the population size is adapted. For example, if in the final stage of testing,  $|S| = 3$  even though the  $p_{preset}$  may be greater than 3, only 3 individuals, each using one of the suspect resources is required for isolation to complete. Such a situation occurs frequently in the beginning of the isolation process. For example, with a resource redundancy ratio,  $rr = 0.5$ , in the first stage, only two individuals are required to cover the entire resource space. Additional individuals will only form tests for resources that are already covered by these two, and will thus be redundant. The number of individuals required in any stage of testing is given by:

$$P_{stage} = \left\lceil \frac{|S|}{n_{share}} \right\rceil \quad (7)$$

Reducing the number of individuals in a test stage provides two benefits. First, it significantly reduces the time required for the fault isolation process. Secondly, it reduces the number of redundant test groups – making the algorithm more reasonable. In particular, a *reasonable* group testing procedure is one that contains no test whose outcome can be predicted from outcomes of other tests conducted previously [Du and Hwang 2000].

Once  $n_{share}$  and  $p_{stage}$  are known, the individuals for a given generation are created, and then tested. As shown Fig. 5, testing comprises the third phase of the isolation process. The tests are by randomly selecting an individual  $\mathbf{c}_i$  for instantiation on the FPGA. A test vector  $t_j$  is then applied to the individual. If  $\mathbf{D}(t_j, \mathbf{c}_i) \neq \{\}$ , all H matrix entries  $H_{(x,y)}$  are incremented by one where  $(x, y)$  are the coordinates of all  $r_i(x,y) \in \mathbf{c}_i$ . Regardless of whether there is a discrepancy, this configuration is then replaced by another, and the testing continues. When a configuration containing the defective resource is tested, the probability of the fault being expressed as a discrepant output is governed by the articulation rate,  $a(c_i)$ , of that configuration. Once a fault is articulated, the set of suspects will be reduced to the intersection of the resources utilized by  $\mathbf{c}_j$  and the resources  $H_{(x,y)} = H_{max}$ . Thus:

$$S_{new} = \mathbf{c}_j \cap \{ r_i(x,y) \mid \forall H_{(x,y)} = h_{max} \} \quad (8)$$

where  $h_{max}$  is the maximal element in the history matrix  $H_{(x,y)}$

### 3.6 Overcoming Stasis during Isolation

A state of *Stasis* is encountered in a stage of the isolation procedure if further tests on configurations comprising the stage are expected to lead to no significant reduction in the

number of suspect resources. By definition 4, stasis occurs when forward progress stalls. Defining a method to overcome stasis is essential to ensure fast fault isolation.

Since the suspect resources were equally shared among the individuals in the population, the maximum possible reduction in  $|S|$  is given by:

$$\frac{|S|}{|S_{new}|} = n_{share} \quad (9)$$

Once  $|S_{new}|$  is obtained, the system is defined to have entered a state of stasis, when further improvements to the defect scouring ratio,  $d(stage)$ , have stalled. Further reduction in  $|S|$ , beyond those described in Eqn. 9 is only possible if there exists another individual in the same generation that also utilizes the defective resource. Since such an individual is not guaranteed to exist and to articulate the fault, stasis is declared after the suspect pool is reduced by the factor shown in Eqn. 9. Stasis can also occur when the individual utilizing the defective resource does not articulate the fault, or does so with a very low articulation rate. Thus, stasis occurs when no discrepant outputs are observed after a fixed number of inputs are applied.

### 3.7 Walkthrough of Isolation Process

As an example of the isolation process, consider a situation where there is one defective resource with the coordinates (1,8) in a set of  $\mathbf{R} = 100$  resources, where  $r_i(x,y) \in \mathbf{R}$ ,  $0 < x,y < 9$ . For simplicity, let us assume that a configuration that utilizes the defective resource always articulates the fault at the output. The number of resources required to implement the application is  $n_{reqd} = 35$ . Thus, for the first stage, by Eqn. 6:

$$n_{share} = 35 - 100 + 100 = 35$$

For  $n_{share} = 35$ , and population preset,  $p_{preset} = 5$ , by Eqn. 7, we have:

$$\text{Population size, } p_{stage} = \left\lceil \frac{100}{35} \right\rceil = 3$$

Thus, in the first stage, there are three configurations, the first,  $\mathbf{c}_0$  uses resources  $r_i(x,y)$  where  $0 < x < 4$ ,  $0 < y < 5$ , i.e., 35 resources with coordinates (0,0) through (3,4) inclusive;  $\mathbf{c}_1$  uses resources with coordinates (3,5) through (6,9) and  $\mathbf{c}_2$  uses resources (7,0) through (9,9) and (0,0) through (0,4) inclusive. Over a period, each of these three configurations are chosen at random and inputs are applied, until a discrepancy is observed. Since the defective resource (1,8) is used by  $\mathbf{c}_0$ , this configuration will articulate the fault. When this occurs, the H matrix entries corresponding to the resources with coordinates (0,0) through (3,4) used by  $\mathbf{c}_0$  will be incremented by one. The set of

suspect resources  $\mathbf{S}$  now has a cardinality of 35, and contains the resources used by  $\mathbf{c}_0$ . After this first discrepant output, the cardinality of  $\overline{\mathbf{S}}$  exceeds the critical cardinality of 35. Also, the prime realization for this experiment is 1, since  $\mathbf{c}_1$  is known fault-free after  $\mathbf{c}_0$  is identified as the discrepant configuration.

As the set of suspects has diminished by a factor equal to  $n_{share}$ , the next stage of configurations is formed. The number of suspects can be divided equally among the members of this new stage, thus, each new configuration will contain  $35/5 = 7$  suspect resources. The rest of the resources to create the 5 configurations are chosen at random from the  $100-35 = 65$  members of  $\overline{\mathbf{S}}$ . Thus, in the second stage of testing,  $\mathbf{c}_0$  will use the suspect resources with coordinates (0,0) through (0,6) and the other configurations will use 7 suspect resources each, in order. The defective resource with coordinates (1,8) will be utilized by configuration  $\mathbf{c}_2$ . In the tests performed in the second stage,  $\mathbf{c}_2$  will articulate a fault, and H matrix entries corresponding to resources with coordinates (1,4) through (2,0) inclusive will be incremented by one.

In the next stage, only four configurations need be created, with the first three configurations utilizing two of the seven suspect resources. This stage will further reduce  $\mathbf{S}$  to two suspect resources. Finally in the last stage of testing, only two configurations will be created, with the first using the resource with coordinates (1,8) and the second utilizing the resource with coordinates (1,9). Tests on these two configurations will finally yield (1,8) as the defective resource. Thus, in four stages, the defective resource will be uniquely identified.

#### 4. EXPERIMENTAL CONFIGURATION WITH XILINX VIRTEX II PRO

The AGT, together with FIAT, implements the controller for autonomous fault handling. As shown in Fig. 7, this controller receives observed feedback and updates the design population across stages. FIAT has been constructed as part of the work presented using the Python programming language to provide APIs to edit resource constraints, introduce stuck-at-faults, and generate post-place-and-route designs.

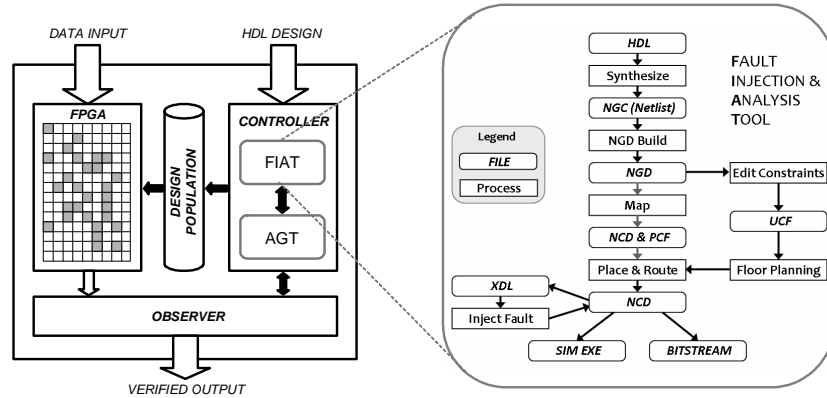


Fig. 7. Fault Isolation Using FIAT

#### 4.1 Creating and Modifying Configurations Using FIAT

Fig. 7 shows the processes that constitute the FIAT design flow. The input files for the process are the HDL files specifying the combinatorial design to be instantiated on the FPGA. These files are synthesized to build a netlist, which FIAT then builds, maps, places and routes using commands provided by the Xilinx ISE 9.1i tools. In the last step a post-place-and-route simulation executable is created using the user-provided testbench and the simulation libraries. The same *Native Circuit Description (NCD)* file used to create the simulation executable can also yield the configuration bitstream for a hardware implementation of the design. The generation of post-place-and-route simulation executables offers a flexible and accurate way of analyzing test routines. In addition to providing methods to implement designs using the Xilinx commands, FIAT provides automated methods to edit physical constraints and to inject faults into configuration bitstreams.

FIAT provides a high-level of control over the physical location of the slices used to create a configuration by providing APIs to modify the *User Constraint File (UCF)*. This enables editing configurations before they are placed and routed. Given a set of suspect resources to be used by each configuration, FIAT creates the UCF files to ensure the use of the suspect resources. It then invokes the Xilinx place-and-route tool provided in the ISE 9.1i platform to realize the designs required by the AGT.

Since it is not viable to destructively modify the FPGA hardware resources, stuck-at faults need to be simulated in the configurations to enable analysis of the AGT algorithm. Stuck-at faults are simulated in the experiments by editing all configurations to exhibit behavior consistent with the presence of a stuck-at fault at one of the input pins of a

specified LUT. To inject the fault, FIAT converts the NCD file, which describes the placed-and-routed design, to the *Xilinx Description Language (XDL)* format using the *xdl* command line tool provided by Xilinx. This text file is then edited to modify the logic function instantiated on the target fault-affected LUT. The presence of a stuck-at fault ties the signal at the input of the fault-affected LUT to zero or one. After fault injection, the XDL file is converted back to an NCD file. Placement and routing is then completed automatically using the Xilinx tools included in the ISE 9.1i suite.

FIAT precludes the need to edit the configuration bitstream directly. Throughout the design flow, the Xilinx 9.1i ISE tools are used for all processes except for those that parse and edit the UCF and XDL files. The Xilinx design tools, such as *netgen*, *par*, *ngdbuild*, and *fuse* are invoked by FIAT in the design flow to place and route the edited designs. This principle of interfering minimally with the functions of the Xilinx ISE reduces accidental bitstream errors that may invalidate the design or irrecoverably damage the FPGA.

#### 4.2 Experimental Design for the Xilinx Virtex-II Pro FPGA

Experiments were conducted on a Virtex-II Pro FPGA xc2vp4-7ff672 model using the Xilinx ISE 9.1i design platform. The 7ff672 package provides 3008 slices and 348 Input-Output Blocks (IOBs).

To analyze performance of the algorithm, the following characteristics are defined by the functionality of the application implemented on the FPGA:

**Definition 6.** The *application resource demand*,  $n_{reqd}$  is the minimal cardinality of any design configuration  $|\mathbf{c}_i|$ , required to implement the application on the FPGA.

**Definition 7.** The *resource redundancy* ratio,  $rr$  is defined as the ratio of the application resource demand to the cardinality of the set of all resources  $|\mathbf{R}|$

$$rr = \frac{n_{reqd}}{|\mathbf{R}|} \quad (2)$$

**Definition 8.** The *critical cardinality* is the cardinality of  $|\mathbf{S}^C|$  such that  $|\mathbf{S}^C| = n_{reqd}$ .

**Definition 9.** The *prime realization* is the index  $i$ , of the first identified subset  $\mathbf{c}_i$ , which satisfies the two conditions:  $\mathbf{c}_i \subset \overline{\mathbf{S}}$  and  $|\mathbf{c}_i| \geq n_{reqd}$ .

The DES-56 implementation utilizes 304 slices and 191 bonded IOBs. Thus, for these experiments, the application resource demand,  $n_{reqd} = 304$ . The total gate-equivalent count for the design is 5266. The area under test on the FPGA can be varied by controlling the total resources,  $\mathbf{R}$ , available for placing and routing the design. This

enables varying  $rr$  for the experiments. Initially, the DES-56 core was synthesized, mapped, placed, and routed on the FPGA. This model was later modified using FIAT according to the requirements of the AGT to form configurations and test stages. For each of these configurations, a simulation executable was created using a testbench. The inputs for the DES-56 circuits were obtained from the National Bureau of Standards publication 500-20 [Gait 1997]. These inputs comprehensively test the functionality of hardware implementations of DES-56. Sixty of these inputs, representing a cross-section of the NBS test suite were used to create the test bench.

## 5. EXPERIMENTAL RESULTS

Experiments were conducted using post-place-and-route designs created for the Xilinx Virtex-II Pro FPGA. A *56-bit Data Encryption Standard (DES-56)* encryption/decryption implementation was used in generating the data.

### 5.1 Isolation Progress Across Test Stages

Fig 8. shows the progress of defect isolation across various stages for  $p_{preset} = 5$  for three different experimental runs. The best performance is seen in experiment 1, where fault isolation is completed using 5 stages of tests. A total of 21 different configurations were created to identify the single defective resource. In the first test stage, three individuals were created, one of which utilized the fault-affected resource and articulated the fault. Thus, at the end of stage 1, the number of suspect resources drops from 625 to 304. The two individuals in stage 1 that do not utilize the defective slice are the prime realizations of the circuit which can provide fault-free implementations on demand. Also, by the end of stage 1,  $|\bar{S}| = 625 - 304 = 314 > n_{reqd}$ , and thus, critical cardinality is met. In stages 2, 3, and 4, five configurations each are created, as  $p_{stage} = p_{preset} = 5$ . Since the equal sharing method is used to create the configurations in each group, the number of suspect

resources decreases by a factor of  $\left\lceil \frac{|S|}{p_{stage}} \right\rceil$  in each stage. In the final stage, since  $|S| = 3$ , only three configurations are created. The number of discrepant outputs in all the tests is equal to the number of test stages since at the occurrence of the first discrepant output, the creation of a new group of configurations is initiated.

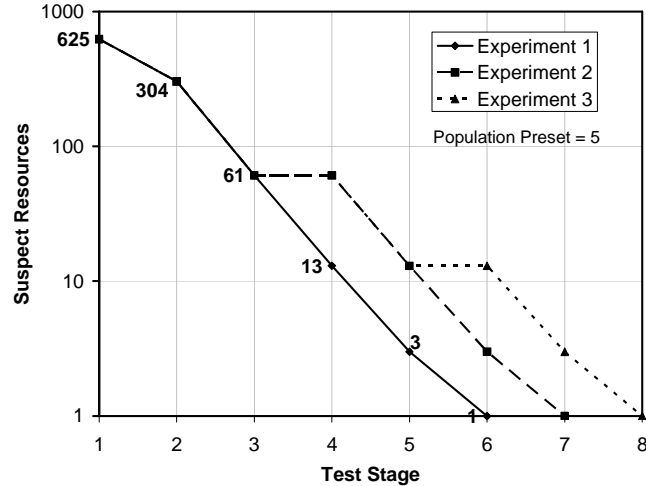


Fig. 8. Fault Isolation Progress Across Stages for  $p_{preset} = 5$

As shown in Fig. 8, in the Experiment 2, no progress is made in the third stage of testing, where the number of suspect resources remains at 61. This is due to the fact that the individual utilizing the fault-affected resource does not articulate the fault, leading to a stasis in the system. In stage 4, five new individuals replace the configurations in the population. In this stage, the configuration with the faulty resource articulates the fault, leading to a decrease in the number of suspects. Similarly, in the third experiment, stasis occurs in the fifth stage. This increases the number of stages to isolate the fault and the total number of configurations created.

In the best performing experiment, five stages were required, and five tests with discrepant outputs are observed before the defect is isolated. Even in the worst case, with a test stage containing configurations that do not articulate the fault, only five discrepancies are observed. Non-articulating individuals that use the faulty resource increase the time taken to scour the defects, but do not affect the observed goodput. In addition, in all these case, since  $rr < 0.5$ , the prime realization, as well as a non-suspect set of resources with a cardinality greater than the critical cardinality are obtained after the first discrepant test output.

## 5.2 Effect of Population Preset on Scouring Rate

The scouring rate is directly proportional to the population preset,  $p_{preset}$ . Table II lists the observed defect scouring performance for varying values of  $p_{preset}$ . A total of 15 experiments were conducted for each value of  $p_{preset}$ . The physical logical resource overhead for the AGT-based technique can be varied by adjusting the resource

redundancy ratio,  $rr$ . In all these experiments, initially,  $|\mathbf{R}| = 625$ . This value was chosen as  $25^2 = 625$  yields a redundancy ratio  $rr = 304/625 = 0.49 \approx 0.5$ . As the column labeled M2 in Table II indicates, throughout the experiments, a subset of non-suspect resources, with cardinality  $\overline{\mathbf{S}} > n_{reqd}$ , is identified after the first stage of testing. Similarly, from the results for metric M3 in Table II, it is shown that the prime realization, which provides a fault-free replacement configuration, is consistently identified from within the first group of configurations. The number of discrepant outputs, or positive tests required to isolate the fault is the same as the number of stages, since the articulation of a fault will immediately improve the scouring rate and trigger formation of the next stage of tests.

Fig. 9 shows the best defect scouring performance of AGT for increasing values of  $p_{preset}$ . Each curve depicts the size of the suspect pool,  $|\mathbf{S}|$ , at the beginning of the test stage depicted on the x-axis. For all values of  $p_{preset}$ , population size,  $p_{stage} = 3$  in the first stage of testing, by Eqn. 7. In all other stages except the last stage,  $p_{stage} = p_{preset}$ . In the last stage,  $p_{stage}$  is equal to the number of remaining suspect resources. The slope of the curve is proportional to the defect scouring ratio, and it increases proportionately with  $p_{preset}$ . Except in the initial and last stages, defect scouring proceeds at a logarithmic rate, when the articulation rate for the configuration utilizing the defective resource is non-zero. Most significantly, across all values of  $p_{preset}$  the defective is isolated with 5 or fewer positive tests. Assuming that the time taken to reconfigure the device is insignificant when compared to the mean time between defects, the AGT-based method can tolerate faults with minimal loss of goodput, with  $p_{preset} = 5$ , which will require the minimal number of reconfigurations.

Table II. Results from 15 Experiments With Varying Population Preset Values

$p_{preset}$	Fault Resolution Metrics*			Number of Stages			Number of Configurations		
	M1	M2	M3	Best	Worst	Mean	Best	Worst	Mean
5	5	1	1	5	7	5.53	21	31	23.67
10	4	1	1	4	5	4.27	27	37	29.67
15	3	1	1	3	4	3.20	35	38	35.47
20	3	1	1	3	4	3.13	39	59	42.73
25	3	1	1	3	4	3.13	41	66	44.27

\* Fault Resolution Metrics:

M1: Number of observed discrepant outputs before the defective resource is isolated.

M2: Number of stages required to surpass critical cardinality for  $\overline{\mathbf{S}}$ .

M3: Number of stages required to identify the prime realization.

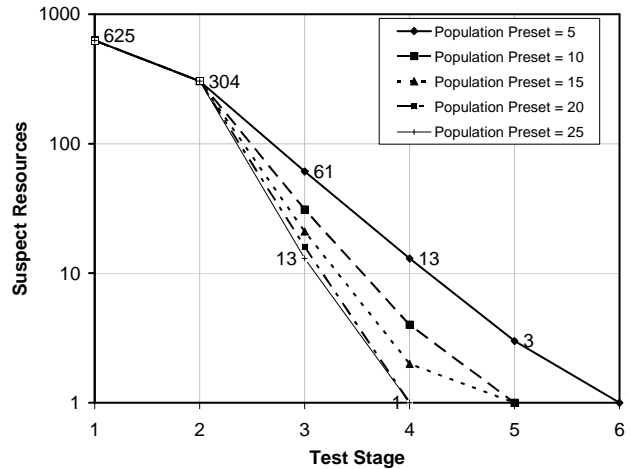


Fig. 9. Effect of Population Preset on the Scouring Rate

The total number of configurations created in each of the five best performing experiments is shown in Fig. 10. As  $p_{preset}$  increases, the total number of configurations increases. However, there is only two extra configurations are required for  $p_{preset} = 25$  as opposed to  $p_{preset} = 20$ . Fig. 10 also shows the number of test stages as a function of  $p_{preset}$ . With increasing  $p_{preset}$ , each stage reduces the number of suspects by a factor proportional to the population size. Thus, with increasing  $p_{presets}$ , though a decreased number of stages are required, the total number of configurations required is increased.

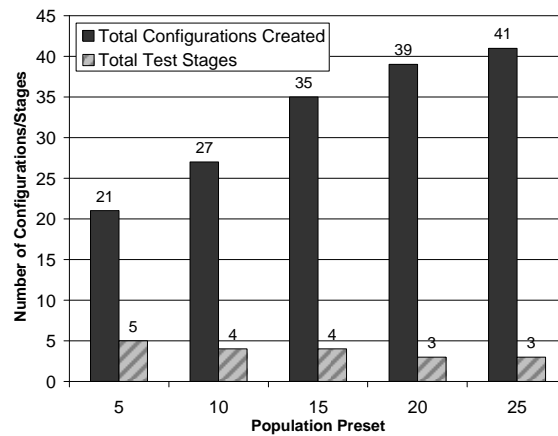


Fig. 10. Total Test Stages and Configurations Created for Varying Population Presets

### 5.3 Maintaining System Goodput During Fault Isolation

System goodput, defined as the percentage of useful outputs, can be maintained at a pre-defined level throughout the fault isolation process using a feedback mechanism and

an observer-controller model. The system goodput decreases each time there is a discrepant output – fault isolation will proceed faster with more frequent discrepancies. Thus, the tradeoff involved in maintaining goodput is that fault isolation will proceed at a slower rate.

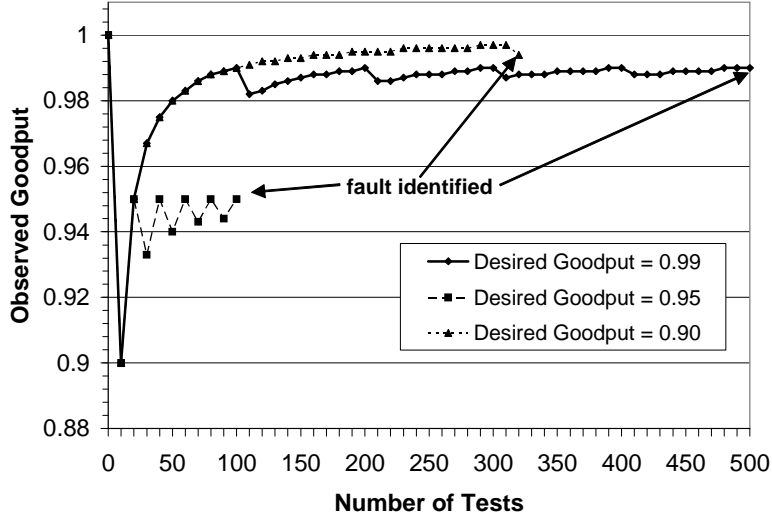


Fig. 11. System Goodput Vs. Total Number of Tests

Fig. 11 shows the observed goodput as a function of the number of tests completed for three different values of required goodput throughout the fault isolation process. In all three experiments, the value used for the population present,  $p_{preset} = 5$ . In the first experiment, the system-level goal is to maintain a goodput of 0.99. A discrepant output is observed in the first ten tests, leading to a goodput of 0.90. Since this is lower than the performance goal, the system responds by utilizing the fault-free configuration until the goodput is restored to 0.99 by the hundredth test. Afterwards, the next stage of testing proceeds. When the fault-affected configuration in the second stage articulates the fault, the goodput drops to 0.982 by the 110<sup>th</sup> test. Again, the system waits for the goodput to return to 0.99 before proceeding with conducting the third stage of tests. After 500 tests, after five positive tests, fault isolation is complete. The observed goodput will then continue to rise past 99%. In the second and third experiments, the goodput requirement is 0.95, and 0.90 respectively. As seen in Fig. 11, for experiment 3, the system goodput never falls below 90% throughout the isolation process. After 10 tests, the goodput falls to 0.90 and rises subsequently until the fault is isolated in 320 tests.

The time taken to create the configurations and reconfigure the FPGA is not reflected in the system goodput measurement. The goodput measured here is that of only the

AGT-controlled system. Since AGT verifies correct functional behavior using output response analysis, it is essential to have an identical fault-free implementation of the same functionality, which would provide the correct outputs to which the outputs of the AGT-monitored configurations can be compared. Under a single-fault assumption, when the portion of the FPGA monitored by the observer is being reconfigured, the system outputs are provided by the other fault-free configuration.

Overall, the AGT-based autonomous method can isolate the single defective with a minimal number of positive tests, as low as 3, as listed in Table II. This result is made even more significant by the fact that this method avoids the use of exhaustive serial test procedures. Of all the previous approaches in Table 1, the roving STARS approach is the only comprehensive fault tolerance solution that isolates defects at a granularity lower than 1% of the total resources on an FPGA. Compared to this approach, the AGT-based technique has a minimal fault detection latency, and thus a higher expected goodput. In addition, as shown by the experiments where the goodput is maintained at a pre-defined value, the AGT algorithm can be used to build an autonomous fault tolerant solution that accomplishes system-level goals.

## 6. CONCLUSIONS

An autonomous group testing-based method to tolerate faults organically on FPGAs is presented. Fast fault isolation using a novel equal sharing group testing method is demonstrated for a single stuck-at fault model. The equal sharing isolation procedure has been developed and its significant characteristics have been analyzed. The FIAT toolkit is developed and demonstrated as a viable solution to inject faults and modify the designs for Xilinx FPGAs. FIAT enables such design modifications and fault simulation using the Xilinx tools provided as part of the Xilinx ISE 9.1i software package.

The adaptive multi-stage group testing algorithm isolates the faulty resources with as few as 3 discrepant outputs in 3 or more stages. Significantly, the AGT-enabled autonomous system achieves fault isolation without using custom exhaustive test procedures. Results are shown for a 5266-gate equivalent DES circuit implementation on a Xilinx Virtex-II FPGA. In all experiments, the approach successfully identifies a known-good set of resources, and the prime realization to replace the fault-affect configuration in the first stage after only one positive test. Additionally, it has been shown that the AGT-controlled system can autonomously meet system-level performance objectives such as maintaining a pre-defined level of goodput.

For future work, the methodology described in this paper with the combination of a modular redundancy scheme such as TMR will address and prevent a wider range of logic resource failures. Further, partial reconfiguration can be exploited to handle some permanent faults adaptively with minimum perturbation to system operation and reduced reconfiguration time. Specifically, existing technology for Xilinx FPGAs allow the division of the resources into *Partial Reconfiguration Regions (PRRs)*. Each PRR has a fixed interface and can support multiple *Partial Reconfiguration Modules (PRMs)*. When a fault is detected in a PRR, the configuration can be replaced with another configuration for that region, provided one exists or can be constructed that does not use the faulty resource.

## ACKNOWLEDGMENTS

We wish to acknowledge the NASA Intelligent Systems for financial support, as well as Mr. John Corbett of Xilinx Inc., for his technical input and feedback.

## REFERENCES

- ABRAMOVICI, M., EMMERT, J., AND STROUD, C. 2001. Roving STARS: an integrated approach to on-line testing, diagnosis, and fault tolerance for FPGAs in adaptive computing systems. In *Proceedings of The Third NASA DoD Workshop*, 73-92.
- BAR-SEVER, Y., BELL, B., DORSEY, A., AND SRINIVASAN, J. 2003. Space Applications of the NASA global differential GPS system. In *JPL Technical Reports*, Institute of Navigation, Portland USA. <http://hdl.handle.net/2014/7840>.
- BARILLOT, E., LACROIX, B., AND COHEN, D. 1991. Theoretical analysis of library screening using an  $n$ -dimensional pooling strategy. *Nucleic Acids Research* 19, 6241-6247.
- BRIDGFORD, B., CARMICHAEL, C., AND TSENG, C.W. 2007. Correcting Single-Event Upsets in Virtex-II Platform FPGA Configuration Memory, *Xilinx Application Note XAPP197*. <http://direct.xilinx.com/bvdocs/appnotes/xapp779.pdf>, Feb. 2007.
- CARMICHAEL, C. 2006. Triple Module Redundancy Design Techniques for Virtex FPGAs, *Xilinx Application Note XAPP197*. <http://www.xilinx.com/bvdocs/appnotes/xapp197.pdf>, July 2006.
- CARMICHAEL, C., CAFFREY, M., AND SALAZAR, A. 2000. Correcting Single-Event Upsets Through Virtex Partial Configuration, *Xilinx Application Note XAPP216*. <http://www.xilinx.com/bvdocs/appnotes/xapp197.pdf>, June 2000.
- CHEATHAM, J. A., EMMERT, J. M., AND BAUMGART, S. 2006. A survey of fault tolerant methodologies for FPGAs. *ACM Transactions on Design and Automation of Electronic Systems* 11, Apr. 2006, 501-533.
- CLANCY, D.J. 2002. NASA challenges in autonomic computing, IBM Almaden Research Center, San Jose, April 2002.
- D'ANGELO, S., METRA, C., PASTORE, S., POGUTZ, A., AND SECHI, G. 1998. Fault-tolerant voting mechanism and recovery scheme for TMR FPGA-based systems. In *Proceedings of IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*.

- DEMARA, R. F., AND SHARMA, C. A. 2005. Self-Checking Fault Detection using Discrepancy Mirrors. In *Proceedings of 2005 International Conference on Parallel and Distributed Processing Techniques and Applications*.
- DORFMAN, B. 1943. The detection of defective members of large population. *Annals of Mathematics and Statistics 14*, 436-440.
- DOUMAR, A., AND ITO, H. 2003. Detecting, diagnosing, and tolerating faults in SRAM-based Field Programmable Gate Arrays: A Survey. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 11, No. 3*, 386-405.
- DU, D., AND HWANG, F. K. 2000. *Combinatorial Group Testing and its Applications*, volume 12 of *Series on Applied Mathematics*. World Scientific.
- FLOYD, S. AND FALL, K. 1999. Promoting the use of end-to-end congestion control in the Internet. *IEEE/ACM Transactions on Networking*, 7, Aug. 1999, 458-472.
- FISCHMAN, M.A., Le, C., AND ROSEN, P.A. 2004. A Digital Beamforming Processor for the Joint DoD/NASA Space Based Radar Mission. In *Proceedings of the 2004 IEEE Radar Conference*, Philadelphia, PA, April 2004.
- GAIT, J. 1977. Validating the correctness of hardware implementations of the NBS Data Encryption Standard. *NBS Special Publ. 500-20*. National Bureau of Standards, Nov. 1977.
- GAREY, M.R., JOHNSON, D.S., AND SO, H.C. 1976. An application of graph coloring to printed circuit testing. *IEEE Transactions in Circuits and Systems 23*, 591-599.
- GARVIE, M., AND THOMPSON, A. 2004. Scrubbing away transients and Jiggling around the permanent: Long survival of FPGA systems through evolutionary self-repair. In *Proceedings of 10th IEEE International On-Line Testing Symposium*, 155-160.
- HUANG, W. AND MCCLUSKEY, E. J. 2001. Column-Based Precompiled Configuration Techniques for FPGA. In *Proceedings of the 9th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, April 29 - May 02, 2001. FCCM. IEEE Computer Society, Washington D.C., 137-146.
- KAHNG, A. B., AND REDA, S. 2006. New and Improved BIST Diagnosis Methods from Combinatorial Group Testing Theory. *IEEE Transactions on Computer-Aided Design*, 25 (3), 533-543.
- KAHNG, A. B., AND REDA, S. 2004. Combinatorial group testing methods for the BIST diagnosis problem. In *Proceedings of the 2004 Conference on Asia South Pacific Design Automation: Electronic Design and Solution Fair* (Yokohama, Japan, January 27 - 30, 2004). with EDA Technofair Design Automation Conference Asia and South Pacific. IEEE Press, Piscataway, NJ, 113-116.
- KATZ, D.S., AND SOME, R. R. 2003. NASA advances robotic space exploration. *IEEE Computer*, 14.1, 52 - 61.
- KASTENSMIDT, F. L., STERPONE, L., CARRO, L., AND REORDA, M. S. On the optimal design of triple modular redundancy logic for SRAM-based FPGAs. In *Proceedings of Design, Automation and Test in Europe*, 2005, 1290 - 1295.
- KEPHART, J. O., AND CHESS, D. M. The vision of autonomic computing. *IEEE Computer*, Vol. 1, 2003, 41-50.
- KEYMEULEN, D., STOICA, A., AND ZEBULUM, R. 2000. Fault-Tolerant Evolvable Hardware using Field Programmable Transistor Arrays, *IEEE Transactions on Reliability*, Vol.49, No. 3.
- KOWALSKI, J.E., GROMOV, K.G., AND KONEFAT, E.H. 2005. FPGA Development for High Altitude Subsonic Parachute Testing. In *Proceedings of the Military and Aerospace Programmable Logic Device International Conference*, Washington D.C. September 2005.
- LACH, J., MANGIONE-SMITH, W.H., AND POTKONJAK, M. 1998. Low Overhead Fault-Tolerant FPGA Systems, *IEEE Transactions on VLSI Systems*, Vol. 6, No. 2, 212 - 321.

- LI, Y., LI, D., AND WANG, Z. 2000. A new approach to detect-mitigate-correct radiation-induced faults for SRAM-based FPGAs in aerospace application. In *Proceedings of the IEE National Aerospace and Electronics Conference*, 588-594.
- MILLIORD, C. J., SHARMA, C. A., AND DEMARA, R. F. 2005. Dynamic Voting Schemes to Enhance Evolutionary Repair in Reconfigurable Logic Devices. In *Proceedings of the International Conference on Reconfigurable Computing and FPGAs*, Puebla City, Mexico, September 28 - 30, 2005, 8.1.1 - 8.1.6.
- MITRA, S., AND MCCLUSKEY, E. J. 2000. Which Concurrent Error Detection Scheme to Choose? In *Proceedings of the International Test Conference 2000*, 985.
- PEROTTI, J.M., LUCENA, A.R., MEDELIUS, P.J., MATA, C.T., BURNS, AND B.M., ECKHOFF, A.J. Advanced Data Acquisition Systems, KSC, Research & Technology Annual Report, 2002.
- RICHTER, U., MNIF, M., BRANKE, J., MÜLLER-SCHLOER, C., AND SCHMECK, H. Towards a generic observer/controller architecture for Organic Computing. In *INFORMATIK 2006 - Informatik für Menschen*, volume P-93 of *GI-Edition -- Lecture Notes in Informatics*, Hochberger C. and R. Liskowsky, Eds. Köllen Verlag, Bonn, Germany, Sept. 2006. 112-119.
- SIEWIOREK, D.P., AND SWARZ, R.S. 1992. *Reliable Computer Systems: Design and Evaluation*, 2<sup>nd</sup> Edition. Digital Press.
- SMITH, C.A.B. 1947. The Counterfeit Coin Problem. *The Mathematical Gazette* 31, 31 -39.
- SROUR, J.R., AND MCGARRITY, J.M. 1988. Radiation effects on microelectronics in space. In *Proceedings of the IEEE Vol. 76, Iss. 11*, 1443 – 1469.
- STERRITT, R. 2005. Autonomic computing. *Innovations in systems and software engineering*, 1(1):79–88, March 2005.
- STROUD, C., and LEE, E., AND ABRAMOVICI, M. 1997. BIST-based Diagnostics of FPGA Logic Blocks. In *Proceedings of the International Test Conference*, 539 – 547.
- TRUSZKOWSKI, W., RASH, J., ROUFF, C., AND HINCHEY, M. 2004. Asteroid exploration with autonomic systems. In *Proceedings of IEEE workshop on the engineering of autonomic systems, at the 11th annual IEEE international conference and workshop on the engineering of computer based systems*, Brno, Czech Republic, 24–27 May. 484–490.
- VIGANDER, S. 2001. Evolutionary fault repair of electronics in space applications. *Masters thesis*, Norwegian University of Science and Technology, Trondheim.
- WYATT, J., SHERWOOD, R., SUE, M., AND SZIJJARTO, J. 1999. Flight validation of on-demand operations: the deep space one beacon monitor operations experiment. In *Proceedings of 5th international symposium on artificial intelligence, robotics and automation in space*, Noordwijk, The Netherlands, June 1999.
- XILINX. 2006. *Virtex-II Pro and Virtex-II Pro X Platform FPGAs: Complete Data Sheet, Version 4.6*. <http://direct.xilinx.com/bvdocs/publications/ds083.pdf>.
- XU, J., SI, P., HUANG, W., AND LOMBARDI, F. 1999. A novel fault tolerant approach for SRAM-based FPGAs, In *Proceedings of the Pacific Rim International Symposium*, 40 - 44.
- YUI, C.C., SWIFT, G.M., AND CARMICHAEL, C. 2003. SEU Mitigation of Xilinx Virtex II FPGAs for Critical Flight Applications. In *Proceedings of the 2003 IEEE Nuclear and Space Radiation Effects Conference*, Monterey, USA.